COAST: The Controller's Assistant

William 11. Duquette

Jet Propulstion Laboratory
California Institute of Technology
Pasadena, California 91109, U.S.A.

ABSTRACT

COAS'J', the Controller's Assistant. is a semi-automated player of the CBS wargame training simulation. It uses rule-based techniques to maneuver a collection of platoons through a complex infiltration mission in response to a single high-1 vel order from a CBS controller.

CDS is driven by orders from controllers of the simulation. It delivers a great deal of internal state information to the controllers' work stations. COAST monitors this information flow, and controls the infiltrating units by sending orders of its own.

Infiltrating units arc controlled by two tiers of rules. Each unit has a plan, or *job-list*, containing the jobs it must successfully complete. Each job has a cluster of job rules which enable the. unit to carry out that job. Over the job rules arc the control rules, which plan the job-list for each unit and handle contingencies.

1 INTRODUCTION

This paper concerns the design of a rule-basczl system to do real-time analysis and control of a military training simulation. The system, COAST, is written in 01'S5 and C. To understand COAST, one must first understand its environment: the Corps Rattle Simulation (CBS).

1.1 The Corps Bathe Simulation

CBS is a wargame/training simulation written in SIMSCRIPT II.5. The training audience consists of brigade.-lcvcl commanders and staffs, and higher echelons up to the corps level. The trainees arc located in command posts as though a real war were in progress, and communicate with their subordinate officers (e.g., battalion commanders) using normal military communications. The subordinates, however, arc playing a role; they arc really operators of the CBS system. They translate the military orders they receive into terms the. simulation can accept.

Between 2.0 and 50 commanders and staffs arc being trained during a typical training exercise; they interact with hundreds of operators, called *controllers*, who in turn interact with CBS. Each has command of one or more simulated military units, to which they send orders via a CBS work station. In return, the simulation sends a wide variety of status reports to the controllers, as shown in Figure 1. Some reports arc displa yed graphically, overlaid on a map of the battlefield. The controllers relay this information back to the training audit.ncc.

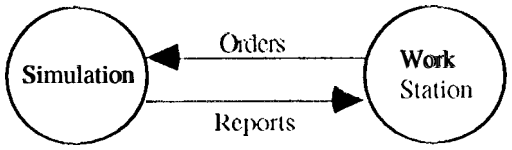

Figure 1: Normal CBS 1 Dataflow

The two conflicting sides in a CBS training exercise arc called BLUFOR (blue force) and 01)101< (opposing force). All trainees are on the BLUFOR side. The OPFOR side is played by a team of specially trained controllers. Running a CBS training exercise is labor-intensive– the overall ratio of controllers to trainees is typically 1:1 or higher, Cost-effectiveness would bc improved if the. ccmtrollcr-[cr-trainee ratio could bc decreased without sacrificing training value.

1.2 An Automated l'layer

For many years, wc have been interested in developing an automated player for CBS. The automated player would assist or replace a human controller, improving the controller-to-trainee ratio. The nominal problem domain is battalion command, since that is the level at which the. controllers operate.. At present, each simulated battalion is manage.ci by crnc or more controllers; in the. heat of battle, as many as four controllers might bc needed. As wc envision it, the player would use rule-based and robotics techniques, and would interact with CBS just as a controller dots: by sending orders and

reading reports. Such a player might allow a single controller m be responsible for several battalions; this would bc a great savings in labor, especially cm the OPFOR sick In addition, an automated player can increase the resolution of the training by playing the battalions as groups of smaller units, such as companies and platoons.

COAST, the Controller's Assistant, is our first step toward an automated player. Written in C and a rule-based language called **01'S5, it** assists a human controller in performing a complex and time-consuming task: battalion infiltration. The essence of infiltration is moving a large number of personnel from one location to another, usually behind enemy lines, while avoiding enemy notice. Since it is difficult for a battalion to move quietly, the battalion generally splits into many smaller units, which move in strict march order along a predetermined route., as shown in Figure 2, infiltration thus trades cohesion and communications for stealth. Once the entire battalion reaches its destination, it reforms- and most likely becomes quilt noticeable in short order. A typical infiltration mission may last for days, and takes place in a small region. COAST automates this process,

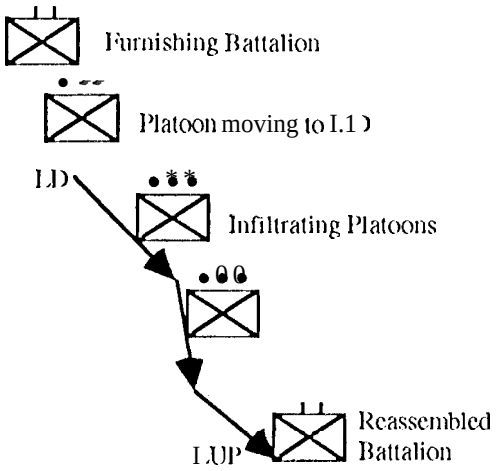

Figure 2: A COAST Infiltration Mission

In response. to an order from a controller, as shown in Figure 3, COAST will take. control of a battalion. The battalion is split into platoons, maneuvered in a strict march order along a mute, and reassembled at the far end. In the meantime, COAST watches for contingency situations, such as enemy activity, and either handles them itself, or requests aid from the controller. With COAST's aid, a controller may start such a mission, and let it proceed with only occasional supervision.

For example, a particular mission might involve infiltrating a battalion, by platoons, from a *line of departure,* to a *link-up point.* First, the controller would send an order to COAST, giving a complete specification of the mission. Next, COAST would split each platoon out of the battalion unit, creating platoon units. Each platoon would be ordered to move to the line-of - departure, and then ordered to infiltrate to the link-up point. As each infiltrating platoon arrived at the link-up point, it would be ordered to link up with the platoons which had already arrived, creating a single, larger unit. When ail of the platoons have merge.d, the battalion has been reassembled at the link-up point. COAST accomplishes this using standard controller orders, and monitors mission progress by reading standard reports.

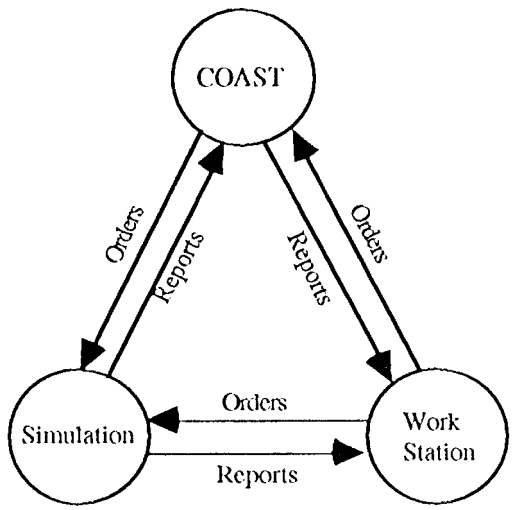

Figure 3: CBS Dataflow with COAST

2 COAST AR CHITECTURE

COAST consists of two parts. The first is the rule-based system, which is called the Mission Operations Manager, or MOM. It is written in OPS5, and is concerned al most completel y with (he. problem domain. The second part, called tile Server, is written in C. The Server is the interface between the MOM and the rest of CBS. The Server receives and processes orders from tile controllers anti reports from the simulation; the Server allows the MOM to send orders to the simulation. One can think of the Server as the MOM's work station. The Server and tile MOM together comprise COAST.

The Server's basic algorithm is as follows:

Receive a controller order
Create a ncw mission
Create a ncw instance of the MOM
Give control of the mission to the MOM

1 .00p
Give new reports to the MOM
Send the MOM's orders, if any,
to the simulation
Until mission termination

In response m the controller's order, COAST creates a new infiltration mission. At the mission start time, the mission specifications and all pertinent battlefield data are loaded into the MOM's working memory, Then, the MOM is give.n control and the loop begins,

The MOM's basic algorithm is as follows:

I .oop
Get reports from the Server
Update battlefield mode]
Do analysis and control
Until mission termination

To behave reasonably, the MOM must always have good knowledge of the simulated battlefield- that is, of tile state of the simulation. Therefore, the MOM maintains a model of the battlefield in its *working memory*. The initial model is loaded into the MOM by the Server with the mission specifications. The MOM updates tile model with each new set of reports.

As each relevant set of reports is received, the. Server loads the ncw data into the MOM's working memory. Next, the MOM uses the data to update its balllc.field model, until it has a complete, consistent picture of the battlefield. Each datum is fit into the structure, and a variety of facts arc inferred. For example, tile MOM constantly tracks the distance of each infiltrating unit from several pertinent locations. Finally, the MOM analyzes the battlefield situation, and decides whether to act. The MOM acts by sending one or more orders to the simulation. The effects of those orders will be learned as future reports arc received. Once the MOM has taken all necessary steps, the loop ends, and the h40M waits for new reports.

1 'or instance, one kind of report COAST receives is a movement report: it says that a particular unit has moved to a ncw location, e.g., unit 1/A/1-75RNG is now at UTM coordinate 33UUQ447365. This report is loaded into working memory, and the model maintenance rule sets compute that 1/A/1-75RNG is within 100 meters of the link-up point. The analysis and control rules determine that 1/A/1 -75RNG can link up with its comrades, and order the simulation to make it so.

The bulk of the processing dc.scribed here is simple in concept, even if complex in implementation, and will not be discussed further. The step labeled "Do Analysis and Control", however, corresponds to the brain of our robot.

3 ANA LYSIS AND CONTROL R ULES

3.1 Jobs and Roles

The previous section described how COAST keeps track of each unit it controls, and noted that it monitors the unit's location, strength, anti so forth. COAST must also keep track of what each unit is trying to do. To facilitate this, wc broke the basic infiltration mission down into a number of *jobs*. Figure 4 shows some of the jobs units might do during a typical infiltration mission, The battalion, pictured at the top, is furnishing platoons to the mission. The platoons must move to the line of departure, where they begin infiltrating, Finally, as the platoons arrive at the link-up point, they must merge with their predecessors. The jobs a unit might do arc determined by its role, For example, the battalion's role is to furnish; it c1ocs not stage, infiltrate, or link-up. An infiltrating platoon will stage., infiltrate, and link-up, but it dots not furnish,

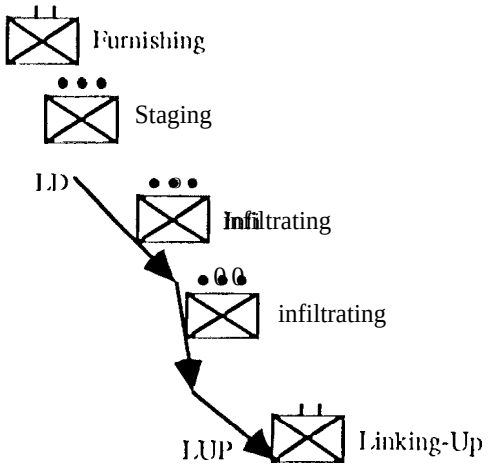

Figure 4: Units and '1'heir Jobs

Each job is define.d by a rule set, which consists of three kinds of rules: progress, success, and failure. Progress rules arc what actually get the job done. Given any reasonable state the unit might be in while doing this job, there must be a progress rule to move it along toward completing the job. Consider a unit with the staging job, If it does not yet exist, it must be split out of the battalion. Once it exists, it must be ordered to move to the line of departure. These statements arc embryonic progress rules.

Success rules recognize that the job has been completed successfully. A unit has staged successfully when it is at the line of departure, the beginning of the

infiltration route. The success rule's only purpose is to identify successful completion.

Failure rules recognize that the job cannot be completed successfully, for jcrb-specific masons. For example, suppose that the platoon cannot be split from the furnishing battalion- perhaps the remaining personnel arc too fcw to staff a new platoon-sized unit. Clearly, the platoon cannot stage, since it camel even be brought into existence. A failure rule would identify the failure, 1 external forces [hat might prevent successful completion will not be recognized by the job's failure rules. Combat, for instance, would prevent the platoon from staging successfully, but handling combat is not part of a staging unit's job description. External conditions arc the province of the control rules, described below.

An infiltrating unit will nominally execute three jobs in sequence: staging, infiltrating, and linking-up. Rather than writing special rules which say, for example, "If a unit has successfully completed staging, it should now begin infiltrating," wc give the unit a *job-list* containing *the* three jobs. As each job is completed successfully, it is removed from the unit's job-list, and the unit proceeds with the next job, This is done by a *control rule.*

3.2 Control Rules

Control rules control units (and thus, the entire mission) by manipulating their job-lists. The most fundamental control rule. is *sequential execution*; as mentioned above: as each job is successfully completed it is remove.d from the, unit's job-list. This allows the next job to begin. There are also initialization control rules, which assign each unit an initial job list based on its role, and contingency control rules, which handle special conditions.

A contingency rule recognizes that the job a unit is doing is no longer appropriate, or that the job's failure flag has been raised. For example, it is inappropriate to continue staging while under attack.

A contingency rule may take any of several actions. First, a job might bc inserted at the beginning of the unit's job-list. For instance, infiltrating units usually hide during the day. When day breaks, a contingency rule puts the DAY-HIDE job at the beginning of the job-list for all units which arc currently infiltrating. When night falls, the DAY-HIDE job succeeds, and the units resume infiltrating- - provided nothing has happened in the meantime.

Next, a unit might be given an entirely new job list. An infiltrating unit which is fired upon too many times is demoralized, and will abort its mission. A contingency rule clears the unit's job list, and gives it a new one which will cause it to move 10 an abort rally point.

In extreme case.s, the contingency rule will transfer the unit to the controller. That is, Cl(lAS'J' will notify the controller that the unit is in trouble, return control of the unit to the controller, and take no further responsibility. This is also the standard operating procedure when COAST encounters problems that are rare or for which solutions have not been implemented.

4 EVALUATION

The combination of control rules and job rules proved an effective way to manage the units in an infiltration mission. Wc believe this technique can bc used, as is, to automate other tasks as well. Nevertheless, there is room for growth.

01'S5, unlike procedural languages, has very little structure. There are only rules, which can work together in convoluted and obscure ways. It is difficult to program a large system at the level of single rules without become hopelessly lost. It is necessary to impose structure on the language, preferably without losing the. fluidity which makes a rule.-bascd language attractive. The development of tbc MOM has been one of increasing structure and expressive power. From the first, wc used several conventional 01'S5 control mechanisms, such as subtasks and agendas. On top of this foundation, wc built the main loop and the. rule sc(s which update the battlefield model. On top of that, wc. built the analysis and control rules.

Originally, the analysis and control rule set was very unstructured. There were no job rules and control rules, and no job-lists, Each unit had a scalar state variable, and control was based on its stale and all external circumstances. For example, a unit infiltrating to the link-up point might have the INFILTRATE-TO-LUP state, A unit that was hiding during the day might have the DAY-HIDE slate.. Essentially, the slate variable was the MOM's summary of the unit's condition and current goal. This proved completely unsatisfactory. Each stale had to have specific rules for each state it might jump to; for example, at day break, INFILTRATE-TO-LUP jumped to DAY-HIDE. '1'0 ensure. that this rule fired when it was supposed to, c.very other INFILTRATE-TO-LUP rule included the condition that it was night time.. Each time a new unit stale was added, wc had to consider all possible. transitions to other states. Finally, a scalar state could not encode enough information. For example, a unit in the DAY-HIDE state forgot whether it had been in the INFIL,'I"RA'J'Il-"1'0-1 .UP state or the. ABORT-TO-RALLY-POINT state., When night fell, it was not clear which state the unit should return to.

The **solution** was, again, increased structure. The individual jobs a unit might do were defined as job rule sets, and the job list was born. Each job rule set is concerned only with that job; all higher-level concerns are the province of the control rules. Increased structure, carefully designed, yielded greater expressive power. Note that the job rule/control rule paradigm relies on the fluidity of the underlying rule-based system. Each unit has its own job, but all units pursue their jobs completely in parallel. Control rules are active at the same time. as job rules, but with higher priority, so that contingencies are handled immediately.

The control rules are still rather unstructured. As COAST's capabilities are expanded, further structure will be needed. The current paradigm allows the MOM to manage a single, focused task. In theory, COAST can be extended to coordinate complex missions comprising a number of interdependent tasks, Each task would control one or more units. In this scheme, COAST would be given mission specifications and resources (e.g., units), and would assign resources to tasks to accomplish the mission. For example, COAST could be ordered to take and hold a particular region. Given appropriate resources, COAST might conduct a company infiltration mission, attack at dawn with the rest of the battalion, and call on an artillery unit to provide fire support.

At present we are enhancing COAST operationally: making it a mm-c user-friendly, useful, and reliable part of the. CBS system. In the future, we hope to extend its capabilities along the lines given above, and to apply the techniques to managing additional kinds of **missions.**

ACKNOWLEDGEMENTS

'This work was sponsored by the U.S. Army Simulation, Training, and Instrumentation Command, through an agreement with the National Aeronautics and Space Administration. Technical guidance was provided by the U.S. Army National Simulation Center. The author thanks Robert G. Chamberlain, Joseph T. Tearey, and Joseph T. Trovato for their comments and assistance, both with this paper and with the development of COAST itself.

AUTHOR BIOGRAPHY

WILLIAM H. DUQUETTE is a Member of the Technical Staff in the Modeling and Artificial Intelligence Applications Group at the Jet Propulsion Laboratory. He received a B.A. degree in mathematics and economics from Claremont McKenna College in 1985, and an M.S. degree in operations research from Stanford University in 1986. He designed and implemented COAST as part of Version 1.4 of the Cbs Battle Simulation,